View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Graph-Based Time–Space Trade-Offs for Approximate Near Neighbors

Thijs Laarhoven
Eindhoven University of Technology
Eindhoven, The Netherlands
mail@thijs.com
https://orcid.org/0000-0002-2369-9067

Abstract

We take a first step towards a rigorous asymptotic analysis of graph-based methods for finding (approximate) nearest neighbors in high-dimensional spaces, by analyzing the complexity of randomized greedy walks on the approximate nearest neighbor graph. For random data sets of size $n = 2^{o(d)}$ on the d-dimensional Euclidean unit sphere, using near neighbor graphs we can provably solve the approximate nearest neighbor problem with approximation factor $c > 1$ in query time $n^{\rho_q+o(1)}$ and space $n^{1+\rho_s+o(1)}$, for arbitrary $\rho_q, \rho_s \geq 0$ satisfying

$$(2c^2-1)\rho_q + 2c^2(c^2-1)\sqrt{\rho_s(1-\rho_s)} \geq c^4. \quad (1)$$

Graph-based near neighbor searching is especially competitive with hash-based methods for small c and near-linear memory, and in this regime the asymptotic scaling of a greedy graph-based search matches optimal hash-based trade-offs of Andoni–Laarhoven–Razenshteyn–Waingarten [5]. We further study how the trade-offs scale when the data set is of size $n = 2^{\Theta(d)}$, and analyze asymptotic complexities when applying these results to lattice sieving.

2012 ACM Subject Classification Theory of computation → Nearest neighbor algorithms

Keywords and phrases approximate nearest neighbor problem, near neighbor graphs, locality-sensitive hashing, locality-sensitive filters, similarity search

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.57

Related Version A full version, containing proofs of all claims and discussing the application of these results to lattice sieving, is available online: `https://arxiv.org/abs/1712.03158`.

Funding This work was supported by ERC consolidator grant 617951.

Acknowledgements The author thanks Marek Eliáš, Jesper Nederlof, and Jorn van der Pol for enlightening discussions and comments on the proofs in the appendices (see the full version).

1 Introduction

Nearest neighbor searching. A key computational problem in various areas of research, such as machine learning, pattern recognition, data compression, and decoding [15, 26, 27, 38, 33, 45], is the *nearest neighbor problem*: given a d-dimensional data set $\mathcal{D} \subset \mathbb{R}^d$ of cardinality n, design a data structure and preprocess $\mathcal{D}$ in an efficient way such that, when later given a query vector $\boldsymbol{q} \in \mathbb{R}^d$, we can quickly find the nearest point to $\boldsymbol{q}$ in $\mathcal{D}$ (e.g. under the Euclidean metric). Since the exact (worst-case) version of this problem suffers from the "curse of dimensionality" [30], a common relaxation of this problem is the *approximate nearest neighbor (ANN) problem*: given that the exact nearest neighbor in the data set $\mathcal{D}$ lies at distance at most r from $\boldsymbol{q}$, design an efficient algorithm that finds an element $\boldsymbol{p} \in \mathcal{D}$

© Thijs Laarhoven;
licensed under Creative Commons License CC-BY

at distance at most $c \cdot r$ from $\boldsymbol{q}$, for a given approximation factor $c > 1$. We will refer to this problem as (c, r)-ANN. Since a naive linear search trivially leads to an $O(d \cdot n)$ time algorithm for solving both the exact and approximate near neighbor problems, the goal is to design a data structure for which queries can be accurately answered in time $n^{\rho+o(1)}$ with $\rho < 1$. Here we will only consider scenarios where d scales with n – for fixed d, it is well-known that one can achieve arbitrarily small query exponents $\rho = o(1)$ [8]. We further assume w.l.o.g. that $n \geq 2^{O(d/\log d)}$ – in case d is larger, we can first perform a random projection to reduce the effective dimensionality of the data set, while maintaining inter-point distances [31].

Partitioning the space. A celebrated technique for efficiently solving ANN in high dimensions is *locality-sensitive hashing (LSH)* [30, 23, 2, 4]. Using hash functions with the property that nearby vectors are more likely to be mapped to the same hash value, one builds several hash tables with buckets containing vectors with the same hash values. Queries $\boldsymbol{q}$ are then processed by computing $h(\boldsymbol{q})$, looking up vectors $\boldsymbol{p} \in \mathcal{D}$ with the same hash value $h(\boldsymbol{p}) = h(\boldsymbol{q})$, and considering these vectors as candidate near neighbors, for each of the precomputed hash tables. Although many hash tables are required to obtain a good recall rate, doing these look-ups in the hash tables is often considerably faster than a linear search through the list.

Whereas LSH requires each point to be mapped to exactly one hash bucket in each hash table, the recent *locality-sensitive filtering (LSF)* [11, 5, 20] relaxes this condition: for each hash filter and each point in the data set, we independently decide whether we add this point to this filter bucket or not. This may mean that some points are added to more filters than others, and filters do not necessarily form a partitioning of the space. Queries are answered by considering filters matching the query, and going through all vectors in these buckets.

For the special case of ANN where the data set lies on the sphere, many efficient partition-based methods are known based on using random hyperplanes [18], regular polytopes [46, 3, 32, 35], and spherical caps [2, 4, 11, 34, 5]. For random data sets of size $n = 2^{o(d)}$, both cross-polytope LSH [3] and spherical LSF [11] achieve the optimal asymptotic scaling of the query complexity for hash-based methods. Spherical LSF further offers optimal time–space trade-offs between the query and update/space complexities [5], while cross-polytope LSH seems to perform better in practice [3, 44, 9]. No optimality results are known for data sets of size $n = 2^{\Theta(d)}$, but spherical LSF outperforms cross-polytope LSH in some regimes [11].

Nearest neighbor graphs. A different approach to the near neighbor problem involves constructing *nearest neighbor graphs*, where vertices correspond to points $\boldsymbol{p} \in \mathcal{D}$ of the data set, and an edge between two vertices indicates that these points are approximate near neighbors [17, 21, 28, 39, 42, 19, 22, 25, 29, 47]. Given a query $\boldsymbol{q}$, one starts at an arbitrary node $\boldsymbol{p} \in \mathcal{D}$, and repeatedly attempts to find vertices $\boldsymbol{p}'$, connected to $\boldsymbol{p}$ in the graph through an edge, which are closer to $\boldsymbol{q}$ than the current candidate near neighbor $\boldsymbol{p}$. When this process terminates, the resulting vector is either a local or a global minimum, i.e. a false positive or the true nearest neighbor to $\boldsymbol{q}$. If the graph is sufficiently well-connected, we may hope to solve the (A)NN problem with high probability in one iteration – otherwise we might start over with a new random $\boldsymbol{p} \in \mathcal{D}$, and hope for success in a reasonable number of attempts.

Compared to LSH and LSF, which often require storing quite some auxiliary data describing the hash tables/filters, and for which the cost of computing hashes or finding appropriate filters is commonly non-negligible, the nearest neighbor graph approach has much less overhead: for each vector $\boldsymbol{p} \in \mathcal{D}$, one only has to store its nearest neighbors in a list, and look-ups require no additional computations besides comparisons between the query $\boldsymbol{q}$ and data points $\boldsymbol{p} \in \mathcal{D}$ in these lists to find nearer neighbors. In practice, graph-based approaches may therefore be more efficient than hash-based methods even if asymptotic analyses suggest otherwise, purely due to the low overhead of graph-based methods.

Besides the standard nearest neighbor graph approach of connecting each vertex to its nearest neighbors [25, 24], various heuristic graph-based methods are further known based on adding connections in the graph between points which are not necessaily near neighbors [43, 37, 16]. This might for instance involve including several long-range, deep links in the graph between distant points, to guarantee that every pair of nodes is connected through a short path [37]. Using hierarchical graphs [36, 16] where vertices are partitioned in several layers further seems to aid the performance of graph-based methods, although again these improvements appear to be purely heuristic.

Comparison of different methods. To find out which method is objectively the best, an obvious approach would be to implement these methods, test them against realistic data sets, and compare their concrete performance. Various libraries with implementations of near neighbor methods are openly available online [14, 44, 40, 24, 16], and recently Aumüller–Bernhardsson–Faithfull presented a thorough comparison of different methods for nearest neighbor searching on various commonly used data sets and distance metrics [13, 9]. Their final conclusions include that the LSH-based `FALCONN` and the graph-based `NMSLib` and `KGraph` currently seem to be the most competitive for their data sets.

Practical comparisons clearly have various drawbacks, as the practical performance often depends on many additional variables, such as specific properties of the data set and the level of optimization of the implementation – better results on certain data sets may not always translate well to other data sets with further optimized implementations. Moreover, some methods may scale better as the dimensionality and size of the data set increases, and it is hard to accurately predict asymptotic behavior from practical experiments.

A second approach for comparing different methods would be to look at their theoretical, proven asymptotic performance as the size of the data set n and the dimension d tend to infinity. Tight bounds on the performance would allow us to extrapolate to arbitrary data sets, but for many algorithms obtaining such tight asymptotic bounds appears challenging. Although for partition-based methods, by now the asymptotic performance seems to be reasonably well understood, graph-based algorithms are mostly based on unproven heuristics, and are not as well understood theoretically. This is particularly disconcerting due to the efficiency of certain graph-based approaches, and so a natural question is: how well do graph-based approaches hold up against partition-based methods in high dimensions? Is it just the low overhead of graph-based methods that allows them to compete with hash-based approaches? Or will these graph-based methods remain competitive even for huge data sets?

Heuristic methods made theoretical. We further remark that in the past, theoretical analyses of heuristic near neighbor methods have not only contributed to a better understanding of these methods, but also to make these methods more practical. Cross-polytope LSH, which is used in `FALCONN` [44], was originally proposed as a fast heuristic method with no proven asymptotic guarantees [46]; only later was it discovered that this method is theoretically superior to other methods as well [3, 32], and can be made even more practical. Recently also for hypercube LSH [46] improved theoretical guarantees were obtained [35], and the heuristic LSH forest approach [10] was also "made theoretical" with provable performance guarantees in high dimensions [7]. The goals of a theoretical analysis of graph-based methods are therefore twofold: to understand their asymptotic performance better, and to find ways to further improve these methods in practice.

1.1 Contributions

We take a first step towards a better theoretical understanding of graph-based approaches for the (approximate) nearest neighbor problem, by rigorously analyzing the asymptotic performance of the basic greedy nearest neighbor graph approach. We show that for random data sets on the Euclidean unit sphere and for arbitrary approximation factors $c > 1$, this method provably achieves asymptotic query exponents $\rho < 1$. We further show how to obtain efficient time–space trade-offs, by making the related near neighbor graph either more connected (more space, better query time) or less connected (less space, worse query time).

Sparse data sets. In the case the data set on the unit sphere[1] has size $n = 2^{o(d)}$, and an unusually near neighbor lies at distance $r = \sqrt{2}/c$, the trade-offs we obtain for finding such a close vector to a random query vector are governed by the following relation.

▶ **Theorem 1** (Time–space trade-offs for sparse data). *For $c > 1$ and $\rho_q, \rho_s \geq 0$ satisfying*

$$(2c^2 - 1)\rho_q + 2c^2(c^2 - 1)\sqrt{\rho_s(1 - \rho_s)} \geq c^4, \tag{2}$$

there exists a graph-based (c, r)-ANN data structure for data sets of size $n = 2^{o(d)}$ on the sphere using space $n^{1+\rho_s+o(1)}$ and query time $n^{\rho_q+o(1)}$.

Minimizing the query complexity in this asymptotic analysis corresponds to setting $\rho_s = \frac{1}{2}$, in which case $\rho_q = c^2/(2c^2 - 1)$. Note that, unlike in various hash-based constructions, there is a natural limit to the maximum space complexity of graph-based approaches: we cannot store more than n neighbors per vertex. Our analysis however suggests that when doing a greedy search in the graph, storing more than $\sqrt{n}$ neighbors per vertex does not further improve the query complexity, compared to starting over at a random new vertex.

To compare the above trade-offs with hash-based results, recall that in [5] the authors obtained hash-based time–space trade-offs defined by the following inequality:

$$c^2\sqrt{\rho_q} + (c^2 - 1)\sqrt{\rho_s} \geq \sqrt{2c^2 - 1}. \tag{3}$$

Although in most cases the optimal[2] hash-based trade-offs from (3) are strictly superior to the graph-based trade-offs from Theorem (1), we remark that in the regime of small approximation factors $c \approx 1$ and near-linear space $\rho_s \approx 0$, i.e. when there is no unusually near neighbor and we wish to use only slightly more space than is required for storing the input list, both inequalities above translate to:

$$\rho_q = 1 - 4(c - 1)\sqrt{\rho_s} \cdot (1 + o(1)). \tag{4}$$

Here $o(1)$ vanishes as $\rho_s \to 0$ and $c \to 1$. So for truly random instances without any planted, unusually near neighbors, and when using a limited amount of memory, graph-based methods are asymptotically equally powerful for finding (approximate) nearest neighbors as optimal hash-based methods. In other words, in this regime graph-based approaches will remain competitive with hash-based methods even when the data sets become very large, and the dimensionality further increases.

---

[1] Note that the near neighbor problem on the Euclidean unit sphere is of particular interest due to the reduction from the near neighbor problem in all of $\mathbb{R}^d$ (under the ℓ_2-norm) to solving the spherical case [6]. Furthermore, using standard reductions the results for the ℓ_2-norm translate to results for $\mathbb{R}^d$ with the ℓ_1-norm as well.

[2] Within a certain probing model, the hash-based trade-offs from (3) were proven to be optimal in [5].

On the negative side, our analysis suggests that when there is an unusually near (planted) neighbor to the query, or when we are able to use much more space than the amount required to store the data set, the best known hash-based approaches are superior to the basic graph-based method analyzed in this paper. This could be caused either by our analysis not being tight, the considered algorithm not being as optimized, or due to graph-based approaches simply not being able to profit as much from such unusual circumstances. Note again that hash-based approaches are able to effectively use very large amounts of space, with a large number of fine-grained hash tables/partitions, whereas graph-based methods seem limited by using at most n^2 space. We therefore conjecture that the worse asymptotic performance for "unusual" problem instances is inherent to graph-based methods.

Dense data sets. For settings where the data set consists of $n = 2^{\Theta(d)}$ uniformly random points from the unit sphere, the asymptotic performance of near neighbor searching depends not only on the distance to the (planted) nearest neighbor, but also on the density of the data set, i.e. the relation between d and n. Motivated by data sets appearing in practice, we concretely analyze the case $(\log n)/d \approx 1/5$ and show that the resulting trade-offs for the exact nearest neighbor problem without planted neighbors are significantly better than hyperplane LSH [18]; comparable to or better than cross-polytope LSH [3, 44] and spherical cap LSH [4]; but slightly worse than spherical LSF [11, 5]. The superior asymptotic performance compared to `FALCONN` for this setting suggests that graph-based approaches will remain competitive with the most practical hash-based approaches, even in very high dimensions.

Future work. Various open problems remain to obtain a better theoretical (and practical) understanding of different near neighbor techniques, in particular related to graph-based approaches. We state some remaining open problems below:

- Although our analysis for "small steps" (see the full version) is tight, we assumed that afterwards we either immediately find the planted nearest neighbor as one of the neighbors in the graph, or we fail and start over from a new random node. This seems rather pessimistic, and perhaps the complexities can be further improved with a tighter analysis, without changing the underlying algorithm or graph construction.
- Other heuristic graph-based approaches appear to perform even better in practice, and an open problem is to rigorously analyze their asymptotic behavior as well.
- A practical drawback of using nearest neighbor graphs is that updating the data structure (in particular: inserting new data points) can be rather costly, especially in comparison with hash-based constructions. To make the data structure both efficient and dynamic, one would ideally obtain better bounds on the insertion complexity as well.
- In hash-based literature, the case of sparse data sets has arguably almost been "solved" with upper bounds matching lower bounds (within a certain model). Is it possible to find similar lower bounds for graph-based near neighbor searching?
- Hash-based and graph-based approaches could be considered complementary solutions to the same problem, as worst-case problem instances for one approach are best-case instances for the other. Can both techniques be combined to obtain even better results?

Outline. The remainder of this paper is organized as follows. In Section 2, we introduce preliminary results and notation. Section 3 describes problem instances considered in this paper. Section 4 analyzes the complexities of finding near neighbors with a greedy graph search, and Sections 5 and 6 consider asymptotics for sparse and dense data sets, respectively.

2 Preliminaries

2.1 Notation

Let $\|\cdot\|$ denote the Euclidean norm, and let $\langle\cdot,\cdot\rangle$ denote the standard dot product. We write $\mathcal{S}^{d-1} = \{\boldsymbol{u} \in \mathbb{R}^d : \|\boldsymbol{u}\| = 1\}$ for the Euclidean unit sphere in $\mathbb{R}^d$. We write $X \sim \chi(\mathcal{X})$ to denote that the random variable X is sampled from the probability distribution χ over the set $\mathcal{X}$, and we write $\mathcal{U}(\mathcal{X})$ for the uniform distribution over $\mathcal{X}$. Given a vector $\boldsymbol{x}$ (written in boldface), we further write $x_i = \langle \boldsymbol{x}, \boldsymbol{e}_i\rangle$ for the ith coordinate of $\boldsymbol{x}$.

We denote directed graphs by $G = (V, A)$ where V denotes the set of vertices, and $A \subseteq V \times V$ denotes the set of directed arcs. A directed graph is called symmetric if $(v_1, v_2) \in A$ iff $(v_2, v_1) \in A$. Symmetric directed graphs can also be viewed as undirected graphs $G = (V, E)$ where edges are unordered subsets of V of size 2.

2.2 Geometry on the sphere

Let $\mathcal{C}_{\boldsymbol{x},\alpha} = \{\boldsymbol{u} \in \mathcal{S}^{d-1} : \langle \boldsymbol{u}, \boldsymbol{x}\rangle \geq \alpha\}$ denote the spherical cap centered at $\boldsymbol{x} \in \mathcal{S}^{d-1}$ of "height" $\alpha \in (0, 1)$, and let $C(\alpha)$ denote its volume relative to the entire unit sphere. Let $\mathcal{W}_{\boldsymbol{x},\alpha,\boldsymbol{y},\beta} = \mathcal{C}_{\boldsymbol{x},\alpha} \cap \mathcal{C}_{\boldsymbol{y},\beta}$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}^{d-1}$ and $\alpha, \beta \in (0, 1)$ denote the intersection of two spherical caps, and let its volume relative to the volume of the unit sphere be denoted $W(\alpha, \beta, \gamma)$, where $\gamma = \langle \boldsymbol{x}, \boldsymbol{y}\rangle$ is the cosine of the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$. We will also call the latter objects *wedges*. The volumes of these objects correspond to probabilities on the sphere as follows:

$$C(\alpha) = \mathbb{P}_{\boldsymbol{X}\sim\mathcal{U}(\mathcal{S}^{d-1})}(X_1 > \alpha), \tag{5}$$

$$W(\alpha, \beta, \gamma) = \mathbb{P}_{\boldsymbol{X}\sim\mathcal{U}(\mathcal{S}^{d-1})}(X_1 > \alpha, X_1\gamma + X_2\sqrt{1-\gamma^2} > \beta). \tag{6}$$

The volumes of spherical caps and wedges can be estimated as follows (see e.g. [11, 5]).

▶ **Lemma 2** (Volume of a spherical cap). *Let $\alpha \in (0, 1)$. Then:*

$$C(\alpha) = d^{\Theta(1)} \cdot (1-\alpha^2)^{d/2}. \tag{7}$$

▶ **Lemma 3** (Volume of a wedge). *Let $\alpha, \beta, \gamma \in (0, 1)$. Then:*

$$W(\alpha, \beta, \gamma) = d^{\Theta(1)} \cdot \begin{cases} \left(\dfrac{1-\alpha^2-\beta^2-\gamma^2+2\alpha\beta\gamma}{1-\gamma^2}\right)^{d/2} & \text{if } 0 < \gamma \leq \min\{\frac{\alpha}{\beta}, \frac{\beta}{\alpha}\}; \\ (1-\alpha^2)^{d/2} & \text{if } \frac{\beta}{\alpha} \leq \gamma < 1; \\ (1-\beta^2)^{d/2} & \text{if } \frac{\alpha}{\beta} \leq \gamma < 1. \end{cases} \tag{8}$$

For the wedge, the volume can alternatively be described (up to polynomial factors in d) as the volume of a spherical cap with height $\delta = \sqrt{\frac{\alpha^2+\beta^2-2\alpha\beta\gamma}{1-\gamma^2}}$. In Figure 1, this δ corresponds to the smallest distance from points in $\mathcal{W}_{\boldsymbol{x},\alpha,\boldsymbol{y},\beta}$ to the origin, i.e. the distance from the intersection of the blue and red lines in this projection of the sphere to the origin. In case $\gamma \geq \frac{\alpha}{\beta}$ with $\alpha \leq \beta$, this distance from the origin becomes $\delta = \beta$ and therefore $W(\alpha, \beta, \gamma) = d^{\Theta(1)} \cdot C(\beta)$. In that case, one essentially has $\mathcal{C}_{\boldsymbol{x},\alpha} \cap \mathcal{C}_{\boldsymbol{y},\beta} \approx \mathcal{C}_{\boldsymbol{y},\beta}$.

We will be using various properties of these wedges, and we state some of them below.

▶ **Lemma 4** (Wedge properties). *For $\alpha, \beta, \gamma \in (0, 1)$ with $\gamma < \min\{\frac{\alpha}{\beta}, \frac{\beta}{\alpha}\}$:*

1. *For d sufficiently large, $W(\alpha, \beta, \gamma)$ is decreasing with α, β and increasing with γ;*
2. *For d sufficiently large, $W(\alpha, \beta, \beta)$ is decreasing with β.*

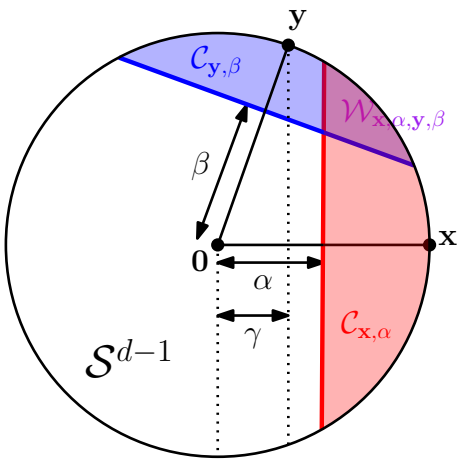

Figure 1 Geometry on the sphere. The relative volume of the wedge, $\mu(\mathcal{W}_{\boldsymbol{x},\alpha,\boldsymbol{y},\beta})/\mu(\mathcal{S}^{d-1})$, is denoted $W(\alpha,\beta,\gamma)$, with γ (as in the sketch) denoting the cosine of the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$.

Proof. For 1. the result follows by taking derivatives w.r.t. α,β,γ of the "asymptotic part" of W (i.e. ignoring the $d^{\Theta(1)}$ term), and observing that these derivatives are always negative, negative, and positive respectively. For 2. we take the derivative w.r.t. β of $W(\alpha,\beta,\beta)$ and observe that this derivative is always negative. ◀

The following lemma further describes that if we add a small amount of "slack" to one of the variables, then the volume of the resulting wedge will still be very similar to the volume of the wedge with the original parameters – if the slack is small, then we only lose at most a polynomial factor in the volume.

▶ **Lemma 5** (Polynomial slack). *For fixed $\alpha,\beta,\gamma \in (0,1)$ with $\gamma < \min\{\frac{\alpha}{\beta},\frac{\beta}{\alpha}\}$ we have*

$$W(\alpha,\beta\pm\tfrac{1}{d},\gamma) = d^{\mp\Theta(1)}\cdot W(\alpha,\beta,\gamma), \qquad \text{and } W(\alpha,\beta,\gamma\pm\tfrac{1}{d}) = d^{\pm\Theta(1)}W(\alpha,\beta,\gamma). \quad (9)$$

Proof. This follows from writing out the left hand sides, pulling out the factors $W(\alpha,\beta,\gamma)$, and noting that the remaining factor $(1\pm\varepsilon_d)^d$ for some expression ε_d is at most polynomial in d, due to the conditions on γ and α,β,γ being constant in d. ◀

3 Random instances

For the graph-based approach in this paper, where points are connected to their nearest neighbor and we perform a walk on this graph starting from a random node, it is impossible to solve worst-case instances of (approximate) nearest neighbor searching.

▶ **Proposition 6** (Worst-case data sets). *For near neighbor graph approaches, where (i) each vertex is only connected to a number of its nearest neighbors, and (ii) queries are answered by performing a greedy search on this graph to obtain better estimates, it is impossible to solve worst-case (approximate) near neighbor instances in sub-linear time.*

Proof. As a potential worst-case problem instance to such a strategy, consider a query $\boldsymbol{q} \approx \boldsymbol{e}_1$, a planted nearest neighbor $\boldsymbol{p}^* \approx \boldsymbol{e}_1$ close to the query, and let all other points $\boldsymbol{p} \in \mathcal{D}\setminus\{\boldsymbol{p}^*\}$ satisfy $\boldsymbol{p} \approx -\boldsymbol{e}_1$. Then the preprocessed near neighbor graph will consist of a large connected component containing the points $\mathcal{D}\setminus\{\boldsymbol{p}^*\}$, and an isolated vertex $\boldsymbol{p}^*$, which may have outgoing edges, but has no mutual friends. With probability $1-1/n$, starting at a

random vertex and performing a walk on this (directed) graph will therefore not yield the true nearest neighbor $\boldsymbol{p}^*$, while all other vertices are only approximate solutions with very high approximation factors; by essentially setting $\boldsymbol{p}^* = \boldsymbol{q}$, we can guarantee that even no reasonable approximate solution will be found. ◀

Although it may be possible to tweak the algorithm and/or the underlying graph so that even such worst-case instances can still be solved efficiently, we will therefore focus on average-case, random instances defined below.

Throughout, we will assume that points $\boldsymbol{p} \in \mathcal{D}$ are independently and uniformly distributed on the unit sphere, except for (potentially) a planted nearest neighbor $\boldsymbol{p}^* \in \mathcal{D}$ which lies very close to the query vector $\boldsymbol{q}$. Alternatively, this can be interpreted as taking a uniformly random $\boldsymbol{p}^* \sim \mathcal{U}(\mathcal{D})$, and choosing the query vector as $\boldsymbol{q} = \boldsymbol{p}^* + \boldsymbol{e}$ for a short error vector $\boldsymbol{e}$. Given such a problem instance, we wish to recover $\boldsymbol{p}^*$ with non-negligible probability.

Note that the near neighbor problem on the Euclidean unit sphere, as considered here, is of special interest as such a solution allows one to the Euclidean near neighbor problem in all of $\mathbb{R}^d$ using techniques of Andoni–Razenshteyn [6]. Furthermore, it is well-known that using standard reductions, the results for the ℓ_2-norm translate to results for $\mathbb{R}^d$ with the ℓ_1-norm as well. Efficient algorithms for the unit sphere therefore translate to solutions for many other problems as well.

For $\boldsymbol{q} \sim \mathcal{U}(\mathcal{D})$, and data sets following a uniformly random distribution, with overwhelming probability the (non-planted) second nearest neighbor $\boldsymbol{p} \in \mathcal{D} \setminus \{\boldsymbol{p}^*\}$ to $\boldsymbol{q}$ has inner product $\langle \boldsymbol{p}, \boldsymbol{q} \rangle = \mu(1 + o(1))$ and lies at distance $\|\boldsymbol{q} - \boldsymbol{p}\| = \sqrt{2(1-\mu)}(1 + o(1))$ from $\boldsymbol{q}$, with μ satisfying:

$$\mu = \sqrt{1 - n^{-2/d}}. \qquad \text{(Alternatively: } n \approx 1/C(\mu).\text{)} \tag{10}$$

The natural scenario to consider for random ANN instances is then to let $c \cdot r = \mu$, so that the single planted nearest neighbor lies at distance $r = \mu/c$ from the target, and so that this planted near neighbor lies a factor c closer to $\boldsymbol{q}$ than all other points in the data set. These problem instances were also studied in for example [3, 5].

Sparse data sets. We will refer to data sets of size $n = 2^{o(d)}$ (in other words: $d = \omega(\log n)$) as *sparse* data sets. For these instances, the expected (non-planted) nearest neighbor distance is $\mu = \sqrt{2} + o(1)$, and the planted nearest neighbor which we wish to recover therefore lies at distance $r = \sqrt{2}/c\,(1 + o(1))$ from the target. Note that the case $n = 2^{o(d/\log d)}$ can always be reduced to $n = 2^{\Theta(d/\log d)}$ through a random projection onto a lower-dimensional space, approximately maintaining all pairwise distances between points in the data set [31].

Dense data sets. We will refer to data sets of size $n = 2^{\Theta(d)}$ ($d = \Theta(\log n)$) as *dense* data sets. In this case, with overwhelming probability the nearest neighbor to a randomly chosen query point $\boldsymbol{q}$ on the sphere will lie at distance $c \cdot r = \mu < \sqrt{2}$ from $\boldsymbol{q}$, and for (c, r)-ANN the planted nearest neighbor would therefore lie at distance $r = \mu/c$. For the limiting case of $c \to 1$, we wish to recover a vector at distance μ. Note that the "curse of dimensionality" [30] does not necessarily apply to average-case dense data sets – finding exact nearest neighbors for random dense data sets can often be done in sub-linear time [34, 11].

4 Graph-based near neighbor searching

4.1 Algorithm description

The nearest neighbor graph and corresponding near neighbor search algorithm we will analyze are very similar to a greedy search in the k-nearest neighbor graph, i.e. very similar to the approach of `KGraph`. Recall that the directed k-nearest neighbor graph $G = (V, A)$ has vertex set $V = \mathcal{D}$, and an arc runs from $\boldsymbol{p}$ to $\boldsymbol{p}'$ iff $\boldsymbol{p}'$ belongs to the k closest points to $\boldsymbol{p}$ in $\mathcal{D}$. Given a query $\boldsymbol{q}$, searching for a nearest vector in this graph is commonly done as outlined in Algorithm 1, with $\varepsilon = 0$, and with $\mathcal{B}(\boldsymbol{p})$ denoting the set of neighbors to $\boldsymbol{p}$ in this graph. Note that for large $k = n^{\Theta(1)}$, this graph is almost symmetric. The condition of belonging to the k nearest neighbors is however somewhat impractical to work with analytically, and so we will use the following slightly different graph (and search algorithm) instead.

▶ **Definition 7** (The α-near neighbor graph)**.** Let $\alpha \in (0, 1)$, and let $\mathcal{D} \subset \mathcal{S}^{d-1}$. We define the α-near neighbor graph as the undirected graph $G = (V, E)$ with vertex set $V = \mathcal{D}$, and with an edge between $\boldsymbol{p}, \boldsymbol{p}' \in E$ if and only if $\langle \boldsymbol{p}, \boldsymbol{p}' \rangle \geq \alpha$.

The parameter α roughly corresponds to (a function of) k: large α correspond to small k, and small α to large k. Asymptotically, the approximate relation between k and α can be stated through the following simple relation $k \approx n \cdot C(\alpha)$. The main difference is that rather than fixing k and varying the required distance between points for an edge, we fix a bound on the distance between two connected points in the graph, and therefore we will have slight variations in the number of neighbors k from vertex to vertex. Notice the similarity with spherical cap LSH [4] and in particular spherical LSF [11, 5]: we essentially use n random spherical filters centered around our data points, and we add vectors to filter buckets the same way as in spherical LSF: if the inner product with the filter vector $\boldsymbol{p}$ is sufficiently large, we add the point to this bucket $\mathcal{B}(\boldsymbol{p})$. However, in the spirit of graph-based approaches we search for a path on the nearest neighbor graph that ends at the nearest neighbor as in Algorithm 1, rather than checking a number of filters close to the query point to see if the nearest neighbor is contained in any of those.

To process a query $\boldsymbol{q}$, we first sample a uniformly random point $\boldsymbol{p} \sim \mathcal{U}(\mathcal{D})$. Then we go through its neighbors $\mathcal{B}(\boldsymbol{p})$ to see if any of these vectors $\boldsymbol{p}' \in \mathcal{B}(\boldsymbol{p})$ are closer to $\boldsymbol{q}$ than $\boldsymbol{p}$. If so, we use this as our new $\boldsymbol{p} \leftarrow \boldsymbol{p}'$, and we again see if any of its neighbors are closer to $\boldsymbol{q}$. We repeat this procedure until no more vectors in $\mathcal{B}(\boldsymbol{p})$ are closer to $\boldsymbol{q}$ than $\boldsymbol{p}$ itself, in which case $\boldsymbol{p}$ is our estimate for the real nearest neighbor $\boldsymbol{p}^*$ to $\boldsymbol{q}$. This may ($\boldsymbol{p} = \boldsymbol{p}^*$) or may not ($\boldsymbol{p} \neq \boldsymbol{p}^*$) actually be the true nearest neighbor to $\boldsymbol{q}$, and to obtain a higher success rate we repeat the process several times, each time starting from a random point $\boldsymbol{p} \sim \mathcal{U}(\mathcal{D})$.

While the focus is on minimizing the query time, Algorithms 2–3 also demonstrate how to do updates to this data structure, when vectors are inserted in $\mathcal{D}$ or removed from $\mathcal{D}$. These parts of the algorithm may well be improved upon, and an important open question is to make updates (in particular insertions) as efficient as partition-based methods, such as in [5].

4.2 High-level proof description

To obtain asymptotic complexities for this approach for the (approximate) nearest neighbor problem, the following statements (for some value γ_{max}) are proven in the full version:

- **Small steps:** If $\boldsymbol{p} \in \mathcal{D}$ satisfies $\langle \boldsymbol{p}, \boldsymbol{q} \rangle \ll \gamma_{max}$, then with non-negligible probability $d^{-\Theta(1)}$ we will find a slightly nearer neighbor $\boldsymbol{p}' \in \mathcal{B}(\boldsymbol{p})$ to $\boldsymbol{q}$.
- **Giant leap:** If $\boldsymbol{p} \in \mathcal{D}$ satisfies $\langle \boldsymbol{p}, \boldsymbol{q} \rangle \approx \gamma_{max}$, the probability of immediately finding the exact nearest neighbor $\boldsymbol{p}^*$ in the bucket $\mathcal{B}(\boldsymbol{p})$ is larger than some given bound.

Algorithm 1 α-NN graph: QUERY($\boldsymbol{q}$).

```
p ~ U(D)        // random starting point
while ⟨p, q⟩ < γ* do        // γ* = ⟨p*, q⟩
    progress ← false
    for each p' ∈ B(p) do
        if ⟨p', q⟩ ≥ ⟨p, q⟩ + 1/d then
            p ← p'        // nearer neighbor
            progress ← true
    if not progress then
        p ~ U(D)        // start over
return p
```

Algorithm 2 α-NN graph: INSERT($\boldsymbol{p}$).

```
B(p) ← ∅        // new bucket
for each p' ∈ D do
    if ⟨p, p'⟩ ≥ α then
        B(p) ← B(p) ∪ {p'}
        B(p') ← B(p') ∪ {p}
```

Algorithm 3 α-NN graph: DELETE($\boldsymbol{p}$).

```
for each p' ∈ B(p) do
    B(p') ← B(p') \ {p}
B(p) ← ∅        // delete bucket
```

Figure 2 Algorithms for querying the α-near neighbor graph with a query point $\boldsymbol{q}$, and for inserting/deleting points from this data structure. Here γ^* is the (expected) inner product between $\boldsymbol{q}$ and its true nearest neighbor. Initializing the data structure is done by choosing $\alpha \in (0, 1)$ and for instance calling INSERT($\boldsymbol{p}$) for all $\boldsymbol{p} \in \mathcal{D}$ to construct the graph adjacency buckets $\mathcal{B}(\boldsymbol{p})$.

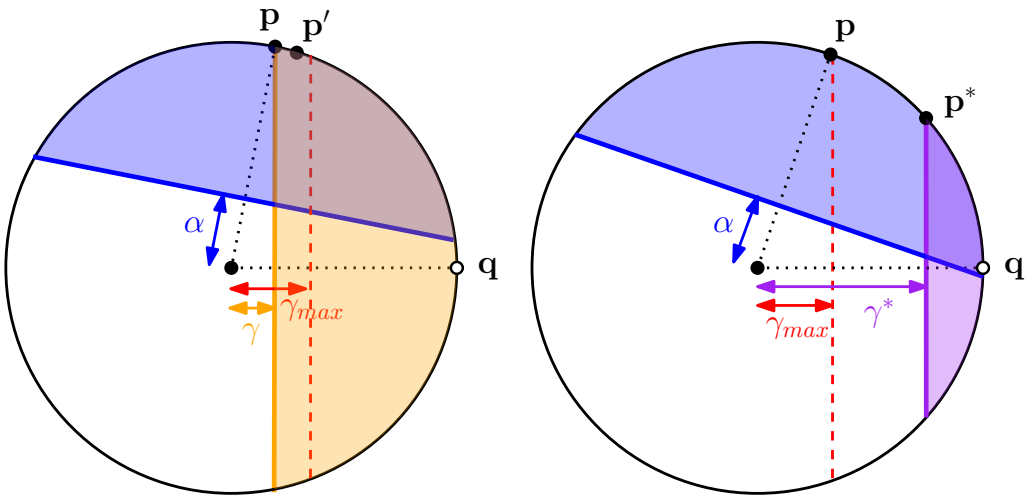

Figure 3 A sketch of the analyses for small steps (left) and the giant leap (right). The point $\boldsymbol{p} \in \mathcal{D}$ denotes the current near neighbor estimate for the query $\boldsymbol{q}$, and we want to make progress either by finding a *slightly nearer* neighbor $\boldsymbol{p}' \in \mathcal{B}(\boldsymbol{p})$ when $\boldsymbol{p}$ is *far away* from $\boldsymbol{q}$ (left), or finding the *exact* nearest neighbor $\boldsymbol{p}^* \in \mathcal{B}(\boldsymbol{p})$ to the query $\boldsymbol{q}$ when $\boldsymbol{p}$ is already quite close to $\boldsymbol{q}$ (right). The threshold separating these two cases is γ_{max}.

- **Small steps (left).** If the current near neighbor $\boldsymbol{p}$ has inner product $\langle \boldsymbol{p}, \boldsymbol{q} \rangle = \gamma \ll \gamma_{max}$ with $\boldsymbol{q}$, then we expect that several points $\boldsymbol{p}' \in \mathcal{D} \setminus \{\boldsymbol{p}^*\}$ still exist which lie (slightly) closer to $\boldsymbol{q}$ than $\boldsymbol{p}$, and we hope at least one of them is an α-near neighbor to $\boldsymbol{p}$ as well. Since a nearer neighbor in $\mathcal{B}(\boldsymbol{p})$ to $\boldsymbol{q}$ by definition lies in $\mathcal{W}_{\boldsymbol{p},\alpha,\boldsymbol{q},\langle \boldsymbol{p},\boldsymbol{q}\rangle}$ (the intersection of the solid spherical caps), and the data set is assumed to be uniformly random on the sphere, the probability of finding at least one such nearer neighbor is proportional to $n \cdot W(\alpha, \gamma, \gamma)$.
- **Giant leap (right).** Once we find a near neighbor $\boldsymbol{p}$ to $\boldsymbol{q}$ with inner product $\langle \boldsymbol{p}, \boldsymbol{q} \rangle \approx \gamma_{max}$, we would like to show that in the next step, we will find $\boldsymbol{p}^* \in \mathcal{B}(\boldsymbol{p})$ with a certain (small) probability. Assuming $\boldsymbol{p}^*$ is uniformly distributed on $\mathcal{C}_{\boldsymbol{q},\gamma^*}$, this corresponds to the probability that $\boldsymbol{p}^* \in \mathcal{W}_{\boldsymbol{p},\gamma_{max},\boldsymbol{q},\gamma^*}$ (the intersection of the solid spherical caps), conditioned on the event $\boldsymbol{p}^* \in \mathcal{C}_{\boldsymbol{q},\gamma^*}$ (the rightmost spherical cap). This probability is therefore proportional to $W(\alpha, \gamma^*, \gamma_{max})/C(\gamma^*)$.

- **Randomization:** Since "giant leaps" may often fail, we argue that starting over at random nodes a sufficiently large number of times leads to a constant success probability.
- **Encountered vertices:** We prove that the number of vertices in each bucket, and on each walk through the graph, can be bounded by small multiples of their expected values.
- **Query complexity:** Using these results, we derive bounds on the time complexity for answering queries, with and without starting over at random nodes in the graph.
- **Space complexity:** Similarly, since the number of edges in the graph can be bounded appropriately, we obtain tight bounds on the required space complexity.
- **Update complexities:** Finally, we analyze what are the (naive) costs for updating the data structure (inserting/deleting points).

Together, these results lead to the following main result, stating exactly what are the costs for finding near neighbors. Unless stated otherwise, the "time" complexity corresponds to the *query* time complexity.

▶ **Theorem 8** (Near neighbor costs). *Using the α-near neighbor graph with $\alpha \in (0,1)$ and $nC(\alpha) \gg 1$, and using Algorithms 1–3, we can solve the planted near neighbor problem with the following costs, with $\gamma_{max} = (1+o(1))\sqrt{\frac{\mu^2-\alpha^2}{1-2\alpha+\mu^2}}$:*

$$\texttt{Time} = d^{O(1)} nC(\alpha)C(\gamma^*)/W(\alpha,\gamma^*,\gamma_{max}), \tag{11}$$

$$\texttt{Space} = O(n^2 C(\alpha)\log n), \tag{12}$$

$$\texttt{Insert} = O(dn), \tag{13}$$

$$\texttt{Delete} = O(nC(\alpha)\log(nC(\alpha))). \tag{14}$$

5 Asymptotics for sparse data sets

Due to space limitations, the derivation of the asymptotics on the various costs of graph-based near neighbor searching for large d can be found in the full version only. Its derivation mainly consists of carefully writing out the costs stated in the previous theorem, and doing series expansions for $\mu = \sqrt{1-n^{-2/d}} = o(1)$.

▶ **Theorem 9** (Complexities for sparse data). *Let $n = 1/C(\mu) = 2^{o(d)}$ and let $\alpha = \kappa \cdot \mu$. Let $\gamma^* = 1 - 1/c^2$ denote the inner product between the query and the (planted) nearest neighbor. Using the α-NNG with $\alpha = \kappa\cdot\mu$ with $\kappa \in (\sqrt{(\gamma^*)^2/(1+(\gamma^*)^2)}, 1)$, with high probability we can solve the sparse near neighbor problem in several iterations with the following complexities:*

$$\texttt{Time} = n^{\left\lceil \frac{1-2\gamma^*\sqrt{\kappa^2(1-\kappa^2)}}{1-(\gamma^*)^2} + o(1) \right\rceil}, \tag{15}$$

$$\texttt{Space} = n^{2-\kappa^2+o(1)}, \tag{16}$$

$$\texttt{Insert} = n^{1+o(1)}, \tag{17}$$

$$\texttt{Delete} = n^{1-\kappa^2+o(1)}. \tag{18}$$

Denoting the query time complexity by $\texttt{Time} = n^{\rho_q+o(1)}$ and the space complexity by $\texttt{Space} = n^{1+\rho_s+o(1)}$, the trade-off between these costs can be expressed using the following inequality:

$$(2c^2-1)\rho_q + 2c^2(c^2-1)\sqrt{\rho_s(1-\rho_s)} \geq c^4. \tag{19}$$

As described in the introduction, for $c \approx 1$ and $\rho_s \to 0$, the above condition on ρ_q scales as $\rho_q \geq 1 + 4(c-1)^2 + O((c-1)^4)$, which is equivalent to the scaling near $c = 1$ for the optimal

partition-based near neighbor method of [5]. For larger c and when using more space, this trade-off is not better than the best hash-based methods – by substituting $\sqrt{\rho_s(1-\rho_s)} \leq \frac{1}{2}$, we obtain the necessary (but not sufficient) condition $\rho_q \geq c^2/(2c^2-1)$, which shows that the query complexity never reduces beyond $\sqrt{n}$, even for large c. This inability to "profit" from large approximation factors may be inherent to graph-based approaches, or at least to the greedy graph-based approach considered in this paper.

For the "balanced" trade-off of $\rho_q = \rho_s = \rho$, the condition on the exponents translates to $\rho \geq c^4/(2c^4 - 2c^2 + 1)$. For small $c \approx 1$, this leads to the asymptotic scaling $\rho = 1 - 4(c-1)^2 + O((c-1)^3)$, which is slightly worse than the optimal hash-based trade-offs of [4, 5]: $\rho = 1/(2c^2-1) = 1 - 4(c-1) + 14(c-1)^2 + O((c-1)^3)$. Also note again the asymptotics of $\rho \to \frac{1}{2}$ for large c for graph-based methods,

6 Asymptotics for dense data sets

For data sets of size $n = 2^{\Theta(d)}$, the asymptotics from the previous section do not apply; these assumed that $\mu = \sqrt{1-n^{-2/d}} = o(1)$. In some applications the relation $d = \Theta(\log n)$ is more accurate, and arguably even if e.g. $d = \Theta(\log n \log\log n)$ grows faster than $\log n$, asymptotics for the sparse regime may be rather optimistic; $\log\log n$ terms are then considered "large", even though for realistic parameters $\log\log n$ may just as well be considered constant.

In the full version, we performed a case study for an application where $\mu = \sqrt{1-n^{-2/d}} = \frac{1}{2}$, so that $n \approx 2^{d/5}$. This example is motivated by algorithms for solving hard lattice problems and cryptanalyzing lattice-based cryptosystems, which have previously been improved using other (hash-based) near neighbor methods [33, 12, 11]. Comparable results may hold for other applications as well; the GloVe data set [41] contains $n = 1.2M \approx 2^{20}$ vectors in $d = 100$ dimensions, which corresponds to $n \approx 2^{d/5}$, while the SIFT features data set [1] has $n = 1M$ vectors in $d = 128$ dimensions, corresponding to $n \approx 2^{0.35d}$. The conclusions regarding the comparison of graph-based near neighbor searching techniques with hash-based methods may therefore apply to such data sets as well.

References

1 Laurent Amsaleg and Hervé Jégou. Datasets for approximate nearest neighbor search, 2010. URL: http://corpus-texmex.irisa.fr/.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

3 Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015. URL: https://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance.

4 Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014. doi:10.1137/1.9781611973402.76.

5 Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, pages 47–66, 2017. doi:10.1137/1.9781611974782.4.

6 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015. doi:10.1145/2746539.2746553.

7 Alexandr Andoni, Ilya Razenshteyn, and Negev Shekel Nosatzki. LSH forest: Practical algorithms made theoretical. In *SODA*, pages 67–78, 2017. URL: `https://dl.acm.org/citation.cfm?id=3039691`.

8 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *SODA*, pages 573–582, 1994. URL: `http://dl.acm.org/citation.cfm?id=314464.314652`.

9 Martin Aumueller, Erik Bernhardsson, and Alexander Faithfull. ANN benchmarks, 2017. URL: `http://sss.projects.itu.dk/ann-benchmarks/`.

10 Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

11 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24, 2016. `doi:10.1137/1.9781611974331.ch2`.

12 Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016. `doi:10.1007/978-3-319-31517-1_1`.

13 Erik Bernhardsson. ANN benchmarks, 2016. URL: `https://github.com/erikbern/ann-benchmarks`.

14 Erik Bernhardsson. ANNOY, 2017. URL: `https://github.com/spotify/annoy`.

15 Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

16 Leonid Boytsov and Bilegsaikhan Naidan. NMSLib, 2017. URL: `https://github.com/searchivarius/nmslib`.

17 M.R. Brito, E.L. Chavez, A.J. Quiroz, and J.E. Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35(1):33–42, 1997.

18 Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. `doi:10.1145/509907.509965`.

19 Jie Chen, Haw-ren Fang, and Yousef Saad. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *Journal of Machine Learning Research*, 10(Sep):1989–2012, 2009. URL: `https://dl.acm.org/citation.cfm?id=1755852`.

20 Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017. `doi:10.1137/1.9781611974782.3`.

21 Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. In *STOC*, pages 609–617, 1997. `doi:10.1145/258533.258655`.

22 Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010. `doi:10.1109/TVCG.2010.9`.

23 Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SOCG*, pages 253–262, 2004. `doi:10.1145/997817.997857`.

24 Wei Dong. KGraph, 2016. URL: `http://www.kgraph.org/`.

25 Wei Dong, Moses Charikar, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011. `doi:10.1145/1963405.1963487`.

26 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, Aug 2010. `doi:10.1109/TIT.2010.2050814`.

27 Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2000.

28 David Eppstein, Michael S. Paterson, and F. Frances Yao. On nearest-neighbor graphs. *Discrete & Computational Geometry*, 17(3):263–282, 1997.

29 Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI*, volume 22, pages 1312–1317, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-222.

30 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. doi:10.1145/276698.276876.

31 William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(1):189–206, 1984. doi:10.1090/conm/026/737400.

32 Christopher Kennedy and Rachel Ward. Fast cross-polytope locality-sensitive hashing. In *ITCS*, pages 53:1–53:16, 2017. doi:10.4230/LIPIcs.ITCS.2017.53.

33 Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015. doi:10.1007/978-3-662-47989-6_1.

34 Thijs Laarhoven. Tradeoffs for nearest neighbors on the sphere. *arXiv*, pages 1–16, 2015. URL: https://arxiv.org/abs/1511.07527.

35 Thijs Laarhoven. Hypercube LSH for approximate near neighbors. In *MFCS*, pages 7:1–7:20, 2017. doi:10.4230/LIPIcs.MFCS.2017.7.

36 Y.A. Malkov and D.A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *arXiv:1603.09320*, pages 1–21, 2016. URL: https://arxiv.org/abs/1603.09320.

37 Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014. doi:10.1016/j.is.2013.10.006.

38 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015. doi:10.1007/978-3-662-46800-5_9.

39 Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, 44(1):1–29, 1997.

40 Marius Muja and David G. Lowe. FLANN, 2013. URL: https://www.cs.ubc.ca/research/flann/.

41 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: http://www.aclweb.org/anthology/D14-1162.

42 Erion Plaku and Lydia E. Kavraki. Distributed computation of the $k$nn graph for large high-dimensional point sets. *Journal of Parallel and Distributed Computing*, 67(3):346–359, 2007.

43 Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor search small world approach. In *ICTA*, 2011. URL: http://www.iiis.org/CDs2011/CD2011IDI/ICTA_2011/Abstract.asp?myurl=CT175ON.pdf.

44 Ilya Razenshteyn and Ludwig Schmidt. FALCONN, 2016. URL: https://falconn-lib.org/.

45 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005.

46 Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.

47 Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-nn graph construction for visual descriptors. In *CVPR*, pages 1106–1113, 2012. doi:10.1109/CVPR.2012.6247790.